Theor Chim Acta (1987) 72: 123–138

Theoretica Chimica Acta

A data-oriented CI program system

F. Sasaki, K. Tanaka, T. Noro, M. Togasi, T. Nomura, M. Sekiya, T. Gonoi and K. Ohno

Department of Chemistry, Faculty of Science, Hokkaido University, Sapporo 060, Japan

(Received July 1, 1986; revised and accepted May 11, 1987)

A program system has been developed for calculation of molecular electronic structure using the configuration interaction (CI) method. Emphasis is placed on the inherent genealogical data structure of the files which a program system produces. Based on this genealogy, a language is provided for users which allows easy and consistent manipulation of files in the new system. Users need only specify a file which contains the desired data, using this file manipulation language. If the desired file does not exist, the new system creates it automatically by calling appropriate modules. The new system may be regarded as a data base equipped with computational ability.

Key words: Program system — CI calculation — Molecular electronic structure — Genealogical data structure — File manipulation language — Data base

1. Introduction

The configuration interaction (CI) method is a most powerful method and its possibilities extend beyond those of the Hartree-Fock approximation. It provides a useful framework for further approximations, such as the perturbational approach.

A CI procedure consists of many steps and produces a number of intermediate data which can or should be used in later steps. For example, atomic integrals which are computed for a certain molecular state need not be recomputed for the calculation of other states, and a common set of configuration state functions (CSFs) should usually be used throughout for different geometries in order to obtain a valid potential surface. Users may want to use various molecular orbitals for a CI calculation, such as SCF orbitals of the ground state, those of excited

states, natural orbitals obtained from a previous CI calculation or a merged set of these orbitals. Therefore, users must save a number of "files" for later calculations. The term "file" is used here merely to denote a cluster of data carrying pertinent information, such as the coefficients of molecular orbitals.

One of the most important problems in designing a program system is how to store and retrieve files. If a program system handles a fixed number of files and their mutual relationships is not altered, the access path can be embedded in the program codes. In this case, storage is divided into appropriate regions either statically or dynamically and each region holds the contents of a particular file. By referring to the same region in the program codes, data can be transferred from one place to another in the program system. The main storage is commonly used for this purpose but the disk storage is also often used when the data space requirement is large.

For a CI program system, the mutual relationship of files is usually much more complex and the method described above may become unsatisfactory. Users may want to attach various combinations of files for each program run. Therefore, a mutual relationship of files cannot be built into the program codes. A simple solution that allows free selection of files is to use an external data set for each file. Access by a DD (data definition) name in the program codes assures local consistency of data transfer among modules, whereas the use of DD statements enables users to attach selected data sets.

The above features might be considered sufficient for a file system, but there are still serious shortcomings. It must be remembered that files produced by a program system are closely related to each other. For example, a CI eigenvector file itself has no meaning unless the CSFs and the molecular orbitals which correspond to the eigenvector are known. A module which processes a CI vector file, e.g. one which creates natural orbitals, will certainly need other files, as mentioned above. Consequently, users must be careful to remember the contents of files and the way in which they are related if they are to be used later, since improper combination of files leads to a meaningless computation. If a program system recognizes the underlying structure of files and provides a systematic means of gaining access to files, file book-keeping will be greatly facilitated. Some of the current program systems in the field of computational chemistry offer data retrieval packages (1, 2) in addition to the standard data-set access method in order to allow more flexible file handling. No attempt, however, has been made to design a quantum chemistry program system which records the complete relationship of the files it produces.

We have designed a CI program system named KAMUY, which automatically records relevant genealogical information when a file is created. By keeping these records in a computer, users have at their grasp the complete relationship of files. In fact, the program system itself utilizes these records for file management. The new CI system provides users with a language for file access. Access can be gained to a particular file in a variety of ways by using this file manipulation language. Users need only specify a file which contains the desired data. If the

desired file does not exist, KAMUY creates it automatically by calling an appropriate module. In other words, users can perform calculations as if they were retrieving data items from a data base.

The system utilizes a virtual storage access method (VSAM) data set (3), which stores both system records and files. The VSAM data set is composed of homogeneous and key accessible VSAM records. A cluster of VSAM records constitutes a KAMUY file.

In Sect. 2, the method of organizing and retrieving files by means of KAMUY is described. Section 3 explains the physical file structure. Concluding remarks are given in the final section. An example of a logging list of a CI calculation for the ground state of the methylene molecule and the syntax of KAMUY input are given in Appendices 1 and 2 respectively.

2. File organization and data retrieval in KAMUY

In this section, we describe how data are organized, generated, and retrieved in the KAMUY system.

2.1. Functional expressions for files

In general, a program system reads in data prepared by users, manipulates them, and finally writes computed results on media. The final results are not produced by a single procedure; a series of procedures has to be invoked, as shown schematically in Fig. 1. R, S, ... and V are "files" as described in the previous section, such as the nuclear coordinates, coefficients of molecular orbitals, etc. The objective of a computation is to generate a file which contains requested data items.

Looking at the relationship shown in Fig. 1, it becomes evident that we can use functional expressions for the specification of a particular file. In Fig. 1, module A produces file T from files R and S. Since the contents of file T are uniquely determined by files R and S, and module A, file T can be expressed as A(R, S). Module A plays the role of a function having two parameters and files R and S

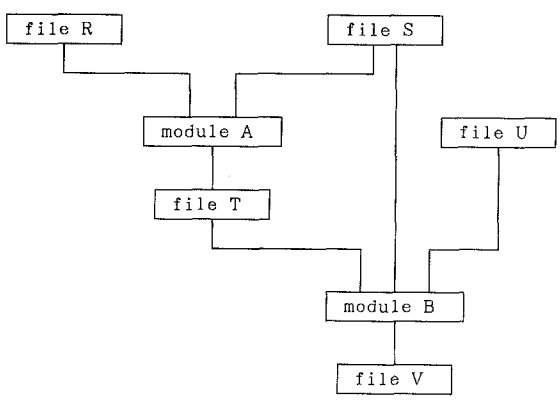

Fig. 1. File production in a computer. Module A produces file T from files R and S, etc

are the actual arguments of the function. Whenever a computational module in the KAMUY system produces a file, it simultaneously creates system records which store the above functional relationship. It should be noted that a module in KAMUY produces only one file at a time. This allows the use of the functional form for the produced file. The organization of all the files based on this file genealogy is of fundamental importance in the system design of KAMUY.

2.2. File manipulation language

The KAMUY system provides a file manipulation language which is based on the relationship of files as described in the previous section. The language is similar in appearance to widely used programming languages, such as FORTRAN or PL/I, but it is essentially different in the following two points.

1. An element of data in the file manipulation language is a file, whereas it is a number or a character string in most programming languages.

2. The return value of a function is a file which is created by a module.

Using this language, users can write a statement

```
MOC = 'MOCL_3B1'
```

which means that a file named "MOCL_3B1" is given the temporary name MOC. In subsequent statements, the symbol MOC can be used as a synonym for the file name. A statement

```
T = TEL (MOC, SOMO, TELSO)
```

means that an output file of the module TEL (transformation module of two-electron integrals) will be given the temporary file name T. The temporary names of the input files of this module are MOC [classification table of molecular orbitals (MO)], SOMO [transformation coefficients from symmetry orbitals (SO) to MO] and TELSO (SO integral file). Users can write such a statement irrespective of whether the corresponding file exists or not, because if the specified file does not exist the system creates it automatically. In fact, writing such statements is actually the only means of activating computational modules of KAMUY.

There follows an example of a list of statements for a sample CI calculation of the methylene molecule.

```
G      = 'GEOM_CH2'
B      = 'BASIS_CH2'
A      = 'AOSPEC_CH2'
TELSO  = JAMTEL(G, B, A)
OELSO  = JAMOEL(G, B, A)
S      = 'SCFSPEC_CH2'
SOMO   = JAMSCF(OELSO, TELSO, S)
CSF    = CSF('CSF_SPEC_3B1')
OELMO  = OEL('MOCL_3B1', SOMO, OELSO, TELSO)
TELMO  = TEL('MOCL_3B1', SOMO, TELSO)
```

```
EIG     = EIG(CSF, TELMO, OELMO)
PRINTF (EIG)
```

Some of the files in the KAMUY system are created by users and are called primary files. The primary files in the above example are "GEOM_CH2" for molecular geometry, "CSF_SPEC_3B1" for a specification of CSFs and so on. These primary files can be prepared either in an interactive mode or in a batch mode. In this example, these primary files are assumed to exist. JAMOEL, JAMTEL and JAMSCF are modules for the generation of one-electron integrals, two-electron integrals and for SCF calculation, respectively. The actual computation represented by these modules is performed by the program system JAMOL3 (4), and these modules merely copy the results of JAMOL3 into the KAMUY system. OEL, TEL, CSF and EIG are modules for the transformation of one- and two-electron integrals, CSF generation and obtaining CI eigenvectors, respectively. Some functions do not return a file as their return value. Instead, they are called solely for their side-effects. PRINTF is one such function; it prints the contents of a file. In Appendix 1, a logging list of a sample run is shown.

In some cases, users may want to find one of the input files from an output file. We define a "role name" and an operator "/" for this purpose. A role name is assigned to each input file of a module to specify the role it plays. For example, module EIG uses three input files: a CSF file, a two-electron MO integral file and a one-electron MO integral file. The role names assigned to them are CSF, TEL_MO and OEL_MO, respectively. By adding the operator "/" and a role name after an expression which represents a file, users can specify one of the input files. In the sample CI calculation, file TELMO can be written as EIG/TEL_MO.

In the file manipulation language of the KAMUY system, any one of the following six expressions can be used to specify a file:

1. File name enclosed by the quote symbol "'"
2. F# followed by a file number assigned by the system
3. Temporary file name
4. Module name (expression, ..., expression)
5. Module name (expression: role name, ..., expression: role name)
6. Expression/role name

Note that the definition is recursive. An expression in the above list can be substituted by any one of the forms.

Use of the file expressions also allows easy and consistent allocation of input files for a module, which is very important. Generation of natural orbitals is taken as an example. Module NAT produces natural orbitals in KAMUY. It requires four input files: (1) a CSF file in the form of distinct row tables (5); (2) a CI eigenvector file; (3) transformation coefficients of molecular orbitals; and (4) a classification table of molecular orbitals. Part of the file relationship is depicted in Fig. 2, where the input files of NAT are marked C, V, S and M. Role names are shown in round brackets. It is apparent that the input files C, S and

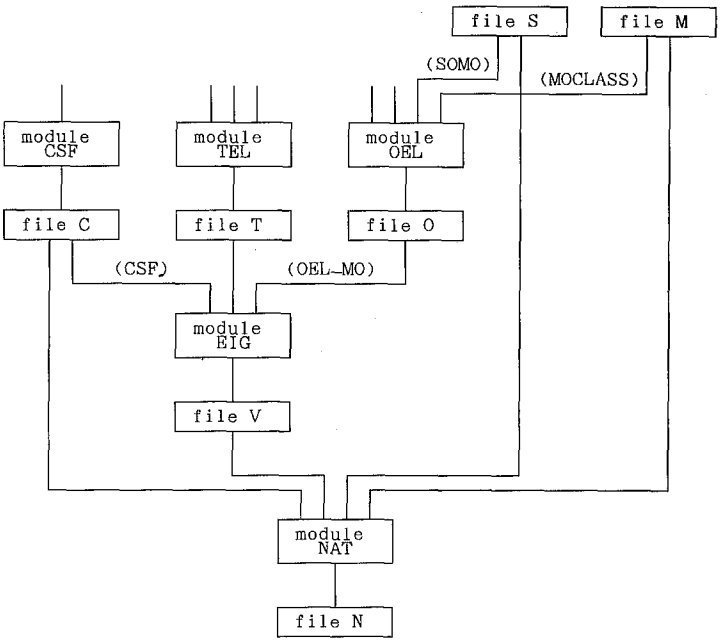

Fig. 2. Tracing input files for module NAT. Three input files C, S and M can be traced back to file V, as shown by thick lines

M can be traced from file V, i.e. file C as V/CSF, S as V/OEL_MO/SOMO and M as V/OEL_MO/MOCLASS. Users, therefore, need only know one file V for the creation of natural orbitals.

KAMUY provides an even simpler method for such access. Users can register in the system a procedural function which defines a desired access path. In the above example, the access path from V to N can be expressed as follows:

```
FUNCTION NATEIG (SOL)
C = SOL / CSF
S = SOL / OEL_MO / SOMO
M = SOL / OEL_MO / MOCLASS
RETURN NAT (C, SOL, S, M).
```

Here, the last statement returns NAT (C, SOL, S, M) as the value of the function NATEIG. Once this function is registered, a statement NATEIG (V) can be written simply to produce natural orbitals. The use of the function NATEIG reduces the burden of file book-keeping for the user and also ensures that the same CSFs, coefficients of molecular orbitals etc, are used for the calculation of both CI eigenvectors and natural orbitals.

2.3. Utilities

Users can activate utility programs in the form of a function call. LISTFL, DESCEND and PRINTF are useful utility programs which are currently incorporated in the KAMUY system. Utility LISTFL lists all files. Utility DESCEND

gives a list of all descendants of a file. It can also be used to erase a file and its descendants. VSAM data space can easily be reclaimed by using this utility. An output list is not produced in KAMUY when a module is activated, since all computed results are stored in files. A utility module PRINTF prints the contents of any file either in the form of a summary or in more detail at any time.

3. File structure in KAMUY

In order to allocate a file, the system should be able to recognize file expressions and to know whether the corresponding file already exists or not. The system, therefore, maintains various system records which describe the logical relationship among generated files. Since the VSAM data set organization is widely available and is capable of random access of records of variable length, we decided to use a VSAM data set to keep system records. Furthermore, the same VSAM data set is used to keep all the logical "files" of KAMUY in order to simplify the designing of the system. Using a VSAM data set as the physical data set has the following advantages:

1. Various access methods are available. Modules in KAMUY can read and/or write data by using keys. The modularity of the program system is greatly enhanced.

2. A data library can easily be incorporated into the system. Appending, update and deletion are made without introducing any change in the program codes.

All VSAM records in KAMUY have a uniform format, as shown in Table 1. The key field (44 bytes) consists of a file number, a data name and data labels. The file number in the key field is the system-assigned logical file number to which the record belongs. The data name and the labels form a unique key within a logical file. The data description field (19 bytes) is used for the manipulation and

Table 1. VSAM record format of KAMUY. All data including system records have this common format. Field lengths are shown in parentheses (unit: byte)

Field	Subfield
Key field (44)	File number (4) Data name (16) Data label 1 (8) Data label 2 (8) Data lable 3 (8)
Data description field (19)	Data pointer (1) Data type (2) Data length (4) Row size (4) Reserved (8)
Data field (variable length)	Data

display of a record. It consisists of a data pointer, a data type, a record size and (for matrix data) a row size. This field serves to increase the locality of data since the contents of the data field can be read without having to resort to any other records or implicit assumptions about the data type or the size. The data field (variable length) is the field for storing the contents of a record.

Each logical file has a file label record, as shown in Table 2. The data name and the labels in the key field for the file label record are binary zeros. The data field of the file label record contains a file name, author, date and time of creation, the location and the computer used, CPU time for generation, restart information, file status codes, and a module number and input file numbers which created the present file. This is one of the fundamental records for the entire file management.

Although the file format is identical, there are four types of files in the system: system files, library files, primary files and secondary files.

The roles of the system files are to map files or module names to file or module numbers assigned by the system, to keep track of the genealogical structure of files, and to register modules, utilities and user-defined functions. Logical file numbers 1–99 are reserved for the system files. System files 1, 2, 5 and 6 are currently used.

System file 1 is used to map file names to logical file numbers. A record in this file has the format as shown in Table 3. The key field consists of a file name and the data field contains the corresponding file number. Inverse mapping of logical file numbers to file names can be done by reading the file label record. System file 2 keeps records on file numbers issued by the system. System file 5 is used to trace a descendant of files. The key field consists of a module number and

Table 2. File label record format of KAMUY. An ancester file is traced by reading this record

Field	Contents
Key field (44)	File number (4) Binary zero (40)
Data description field	Character type
Data field (399)	File name (16) Author (16) Date and time of creation (15) Location (16) Computer used (16) CPU time for generation (16) File status code (8) Restart information (16) Activated module number (4) Input file numbers (36) Comments (240)

Table 3. Record format of file 1. Records are used to map file names to file numbers

Field or subfield	Contents
File number (4)	Binary 1
Data name (16)	File name
Data labels (24)	Blanks
Data description field	Integer type
Data field (4)	Corresponding file number

input file numbers, and the data field contains the resulting output file number (see Table 4). Whenever the KAMUY system analyses the genealogical relationship and identifies a file by its producing module and input files, the system creates a key which corresponds to the specified file. Then, the system searches for a record by using this key in system file 5. If the record exists, it means that the specified file exists. If not, the system allocates the input files and activates the corresponding module to create the specified file. System file 6 is used to map module names to module numbers and to identify subroutines to be called. The KAMUY system compiles and executes statements written in the file manipulation language just as any language processor does. Compiled object codes of defining statements for modules, utilities and function programs are also stored in this system file. Record formats of system file 6 are shown in Table 5.

Library files store various data, such as AO basis sets, geometries of molecules, molecular point group multiplication tables, etc. Since these data are accessed by their keys, modules can read these library data quite easily. Enlargement or update of library files can be done without introducing any change in the program codes.

Primary files are created by users and read by modules. They play the very important role of man-machine interface. In KAMUY, a primary file is created

Table 4. Record format of file 5. Records are used to trace descendants

Field or subfield	Contents
File number (4)	Binary 5
Data name and labels (40)	Module number (4) Input file 1–9 (36)
Data description field	Integer type
Data field (4)	Output file number

Table 5. Record format of file 6. Users input written in the file manipulation language is processed by reading these records. The first type of record is used to map module names to module numbers. The second type contains decoded object program of module, utility and function program definitions. The last type of record stores function labels

Module number to module name record	
Field or subfield	Contents
File number (4)	Binary 6
Data name (16)	####MO
Data lables (24)	Module number
Data description field	Character type
Data field (16)	Module name
Function code record	
Field or subfield	Contents
File number (4)	Binary 6
Data name (16)	Function name
Data lables (24)	CODE
Data description filed	Character type (matrix)
Data filed (24*n)	Compiled code of a module, a utility or a program
Function label record	
Field or subfield	Contents
File number (4)	Binary 6
Data name (16)	Function name
Data lables (24)	LABEL
Data description filed	Character type
Data field (304)	Date of registration and comments

by using a command CREATE. Input format is designed to be as simple as possible. An example of the use of CREATE is shown in Appendix 1. A prompter analyses users' input, warns if inconsistency among input data is found and converts the user's input to the standard record format of KAMUY. In some cases, the prompter reads and shows related files for a further check of consistency. Primary files can be created in a batch mode as well.

Secondary files are those created by modules. The system writes or updates system records and file label records when these files are created or deleted, and maintenance of these records by users is not necessary.

4. Concluding remarks

In this section, we summarize the main features of the new CI system KAMUY.

1. The complete genealogical relationship of files is recorded in the system. Users can easily search for a file using file expressions. There is no distincton between searching for an existing file and creating a new file. The act of searching for a file which has never been accessed induces computation.

2. The system offers programming facilities for file access. The burden on the user from consistent allocation of files and invoking proper modules is greatly reduced.

3. Data records are self-descriptive and accessed by their keys. A programmer only has to know the data names and labels in order to be able to utilize data produced by other modules; a detailed knowledge of the record structure is not required. The same applies to users, who can look at the data on any file without even knowing the data names. It should be noted that the modularity is greatly increased since records are accessed by their keys. There is no implicit passage of information in record format from one module to another.

4. Creation of a primary file can be done interactively. The system gives prompting and checks data whenever possible. The chance of introducing errors in this stage is reduced.

5. Users can see or print the contents of a file in the form of a summary or in more detail at any time.

Program systems easily reach the point where enlarging or updating becomes a formidable task, because the difficulties of data management and the data interface increase quite rapidly. Therefore, data management and handling are becoming the central problems in any large-scale program system. The idea behind designing a new CI program was to solve such difficulties. The starting point of KAMUY was to reveal the inherent data structure, which has not been widely done to date. In this sense, the system may be characterized as data-oriented.

We believe that the system presented here has solved the problems mentioned above to a considerable extent and that it can be adapted to future needs.

Acknowledgements. This work was supported by a Grant-in-Aid for Scientific Research from the Japanese Ministry of Education under project No. 00547012. The computer facilities were provided by Hokkaido University and the Institute for Molecular Science; computer time made available at the computing centers is gratefully acknowledged.

Appendix 1: Logging list of a sample calculation

As a sample calculation, we performed a single and double excitation CI for the ground state of the methylene molecule with a double zeta basis set. The reference function is $1a_1^2 2a_1^2 1b_2^2 3a_1^1 1b_1^1$. The calculation was carried out in a TSS mode on a HITAC M680 computer at Hokkaido University Computing Center. In the following logging list, one- and two-electron SO integrals and molecular orbitals

were generated by the program JAMOL3 and copied into the KAMUY system. The respective files are referred to by the temporary names OELSO, TELSO and SOMO. The user's responses are underlined.

```
KAUMY: CREATE

——→ PRIMARY FILE CREATION STARTS ——→
WHAT KIND OF PRIMARY FILE(S) DO YOU WANT TO GENERATE?
STANDARD TYPE:  ENTER ( JAM / CI / (HELP) ) : CI

⟨⟨MOLECULAR DATA⟩⟩
ENTER MOLECULAR SYMMETRY : C2V

ENTER #ELECTRON : 8

ENTER SPIN ( 2S+1) : 3

ENTER STATE SPACE SYMMETRY : B1

INPUT OF MO SET SIZE
ENTER # OF MO IN EACH REPRESENTATION : ( 1 TO 4 )
8 0 2 4

SD-CI FROM A REFERENCE FUNCTION ? (Y/N) : Y

ENTER CLOSED SHELL MO
(EXCEPT FROZEN & 1-HOLE CORES IN REFERENCES) :

1-2A1 1B2 )

ENTER OPEN SHELL MO IN REFERENCE OR COMPLETE ACTIVE
MO :
3A1 1B1 )

KEY IN THE FILE NAME OF PRIM FILE FOR CSF
F@=: CSF_SPEC_3B1

KEY IN THE FILE NAME OF PRIM FILE FOR TRANSFORMATION
F@=: MOCL_3B1

PRIMARY FILE FOR MOCL HAS BEEN CREATED SUCCESSFULLY
GENERATED PRIMARY FILES
FILE TYPE       F#      F@
CSF SPEC . . .  119     CSF_SPEC_3B1
MO CLASS . . .  120     MOCL_3B1  . . . . .  primary files for CSF and
                                             integral transformation
                                             have been created.
←—— PRIMARY FILE CREATION ENDED ←——

KAMUY: CSF=CSF('CSF_SPEC_3B1');

                                  . . . . .  generation of CSF
——→ M CSF CALLED AT 87-01-28 10:59 CHILD# = 121
←—— M CSF      EXIT AT 87-01-28 10:59 CHILD# = 121 CPU = 0.242
```

```
KAMUY: OELMO=OEL('MOCL_3B1', SOMO, OELSO, TELSO);
                                        ..... integral transformation
                                              of one-electron integrals
——→ M OEL CALLED AT 87-01-28 10:59 CHILD# = 122
←—— M OEL      EXIT AT 87-01-28 10:59 CHILD# = 122 CPU = 0.071
KAMUY: TELMO=TEL('MOCL_3B1', SOMO, TELSO);
                                        ..... integral transformation
                                              of two-electron integrals
——→ M TEL CALLED AT 87-01-28 11:00 CHILD# = 123
←—— M TEL      EXIT AT 87-01-28 11:01 CHILD# = 123 CPU - 1.591
KAMUY: EIG=EIG(CSF, TELMO, OELMO);
                                        ..... get an eigenvalue and
                                              eigenvector
——→ M EIG CALLED AT 87-01-28 11:02 CHILD# = 124
←—— M EIG      EXIT AT 87-01-28 11:08 CHILD# = 124 CPU = 13.255
KAMUY: PRINTF(EIG);
                                        ..... print contents of file EIG
——→ U PRINTF   CALLED AT 87-01-28 11:09
***** CI SUMMARY   ENERGY & COEF OF CONF. *****
SYMMETRY POINT GROUP   C2V
CSF SYMMETRY           B1
2S+1                    3
# OF CSFS              945
SEQUENCE OF VALENCE ORBITALS
( INTERNAL MO# AND REPRESENTATION )
1A1 2A1 1B2 3A1 1B1 4A1 5A1 6A1 7A1 8A1 2B1 2B2 3B2 4B2
.....
.....
EIGEN_VALUE OF    1-ST STATE          -38.994836 HARTREE
--- LIST OF COEFFICIENTS AND CONFIGURATIONS ---
   CSF#            CORE  EXT.      VALENCE PART COUPLING
      1  0.980719  ----  ----
                                   F F F U U E E E E E E E E
.....
.....
--- LIST OF ENERGY LOWERING AND CONFIGURATIONS ---
   CSF#            CORE  EXT.      VALENCE PART COUPLING
    346 -0.009595  ----  ----     E F F U U E E E E F E E E E
    755 -0.003986  ----  ----     F F E U U E E E E E E F E
    739 -0.003153  ----  ----     F F U U E E E E E U E D E
.....
.....
←—— U PRINTF   EXIT   AT 87-01-28 11:09
```

```
KAMUY: DESCEND(SOMO);

                                                .....  print descendant list
                                                       of file SOMO
——→ U DESCEND   CALLED AT 87-01-28 11:11
.....                                           .....  print of file label
***** GENEALOGICAL TREES *****

      122 ←—— OEL
         124 ←—— EIG
      123 ←—— TEL

←—— U DESCEND    EXIT   AT 87-01-28 11:11
KAMUY: END                                      .....  TSS session end
```

Appendix 2: Syntax of KAMUY

We describe the syntax of KAMUY input using the following rules. Text enclosed in brackets ⟨ and ⟩ represents unit names of the syntax and the symbol ::= separates a unit name on the left side from its definition on the right. Items separated by the symbol | and enclosed in square brackets [and] represent a list of alternatives. The outermost square brackets are omitted. We freely use an ellipsis ..... or text enclosed by the symbols " " in cases where the exact description of the syntax is not essential. Other characters should be written as they are expressed on an input of KAMUY.

```
⟨KAMUY input⟩      ::= END
                     | ⟨input element⟩⟨KAMUY input⟩

⟨input element⟩    ::= ⟨statement⟩
                     | ⟨function subprogram⟩
                     | ⟨registration of function subprogram⟩
                     | ⟨primary file creation utility⟩
                     | ⟨WITH utility⟩
                     | ⟨PRINT utility⟩
                     | ⟨DEBUG utility⟩
                     | ⟨* utility⟩

⟨statement⟩        ::= ;                "null statement"
                     | ⟨expression⟩
                     | ⟨temporary file name⟩ = ⟨expression⟩
                     | EVAL ⟨expression⟩, ..... ⟨expression⟩

⟨expression⟩       ::= ⟨factor⟩
                     | ⟨expression⟩ / ⟨role name⟩
```

```
⟨factor⟩                ::= [ FILE# | F# ] ⟨file number⟩
                          | '⟨file name⟩'
                          | ⟨temporary file name⟩
                          | ( ⟨expression⟩ )
                          | ⟨function name⟩ ( ⟨parameter list⟩ )

⟨parameter list⟩        ::= ⟨parameter⟩
                          | ⟨parameter list⟩, ⟨parameter⟩

⟨parameter⟩             ::= ⟨expression⟩
                          | ⟨expression⟩: ⟨role name⟩

⟨temporary file name⟩, ⟨function name⟩ or ⟨role name⟩
                        ::= "an identifier"

⟨file name⟩             ::= "a character string"

⟨file number⟩           ::= "an unsigned integer"

⟨registration of function⟩
                        ::=[ REGISTER | REG ] ⟨function name⟩ .....

⟨function subprogram⟩
                        ::=[ FUNCTION | FUNC ] ⟨function name⟩
                          ( ⟨role name⟩, ..... ⟨role name⟩ )
                          [ MODULE "for module definition"
                          | UTILITY "for utility definition"
                          | ⟨statement⟩ ..... ⟨statement⟩
                           RETURN ⟨expression⟩
                                        "for function program" ]

⟨primary file creation utility⟩
                        ::= CREATE [ JAM | CI | GENERAL ] .....
                          "prompting messages are issued for each
                           type of primary file (JAM for SCF related
                           files, CI for CI related files and GENERAL
                           for any type of primary file)"

⟨WITH utility⟩          ::= WITH ⟨expression⟩
                          [ READ | WRITE | APPEND | UPDATE | DELETE
                              "record manipulation command"
                          | READFL | UPDATEFL
                              "file label manipulation command"
                          | APPENDF@ | UPDATEF@
                              "file name manipulation command"
                          | DELETFB ] .....
                          "This command deletes all data parts of a
                           file except for the file label in order
                           to save space of the VSAM data set."
```

⟨DEBUG utility⟩ ::= DEBUG .....
"This command sets or clears debugging flags for an individual subroutine or a group of subroutines."

⟨* utilities⟩ ::= *VEDIT .....
"activate a utility program for the VSAM data set handling"
| *GPSET | *GPREAD | *GPDEL | *GPLIST
"set, read, delete or list the values of global parameters (Global parameters are used for session control)"

Other features

1. BACK, QUIT or attention key cancels an input: they can be used to correct a keyed-in datum.

2. HELP returns a help message.

3. PROMPT ON or OFF activates or deactivates the issuing of prompting messages.

4. Users can issue a TSS command in a session. A text followed by the symbol $$$ is regarded as a TSS command.

5. KAMUY input texts can be switched from one data set to another. The symbol $$# followed by a data set name indicates that the successive input texts are to be taken from the specified data set.

References and notes

1. Kashiwagi H (1974) Center News of Hokkaido University Computing Center 6 10–23 (in Japanese)
2. Diercksen GHF (1982) Comput Phys Commun 25: 1–6
3. OS/VS Virtual Storage Access Method (VSAM) Programmer's Guide GC26-3838-3, IBM Corporation (1978)
4. JAMOL3 is registered as one of the library programs at the Computing Center of the Institute for Molecular Science
5. Shavitt I (1977) Int J Quantum Chem 11S: 131–148